FTML practical session 8: 2023/30/24

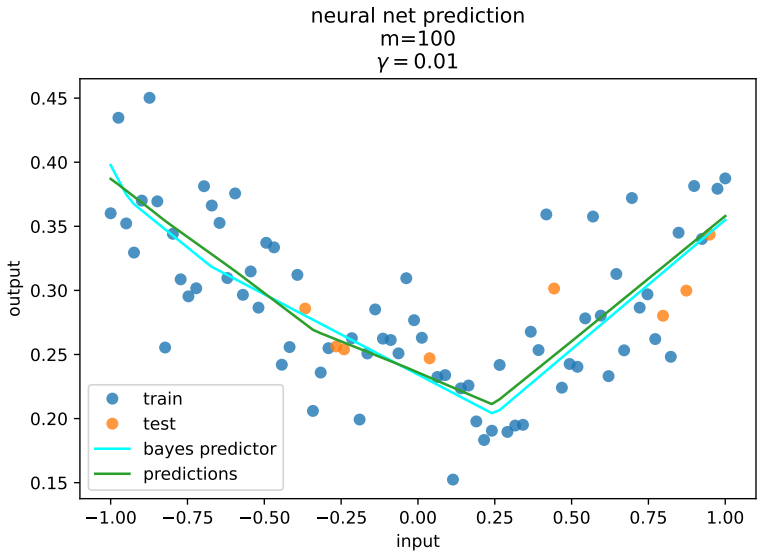


TABLE DES MATIÈRES

1 DENSITY ESTIMATION WITH GAUSSIAN MIXTURES

In this exercise, we will learn a probability distributions from a dataset (density estimation). We use Gaussian mixtures (GMM) as parametric models.

1.1 Gaussian mixtures

Gaussian mixtures model the considered data by a number p of Gaussian distributions. Each one of the p subdistributions is called a "component".

https://fr.wikipedia.org/wiki/Mod%C3%A8le_de_m%C3%A9lange_gaussien

https://scikit-learn.org/stable/modules/mixture.html

You can read and run the file **example_2D.py**, that generates a dataset in 2D, and learns Gaussian mixtures models for different numbers of components. The results are displayed in figures 1, 2, 3, 4, and 5.

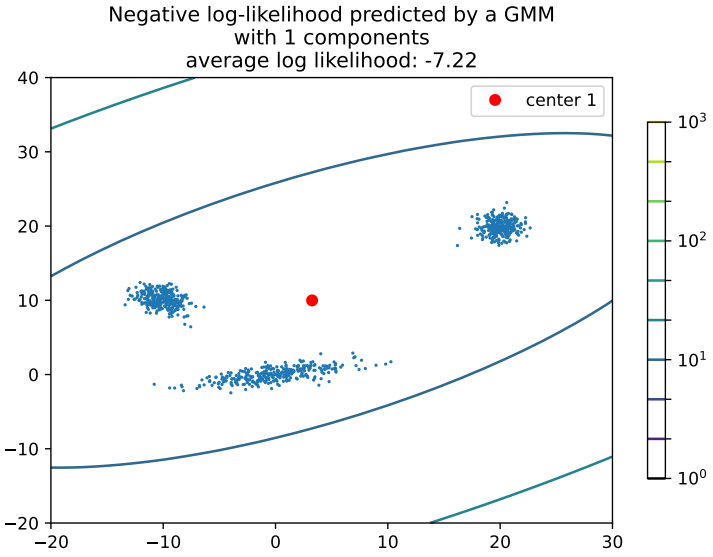


FIGURE 1 – Gaussian mixture with 1 component.

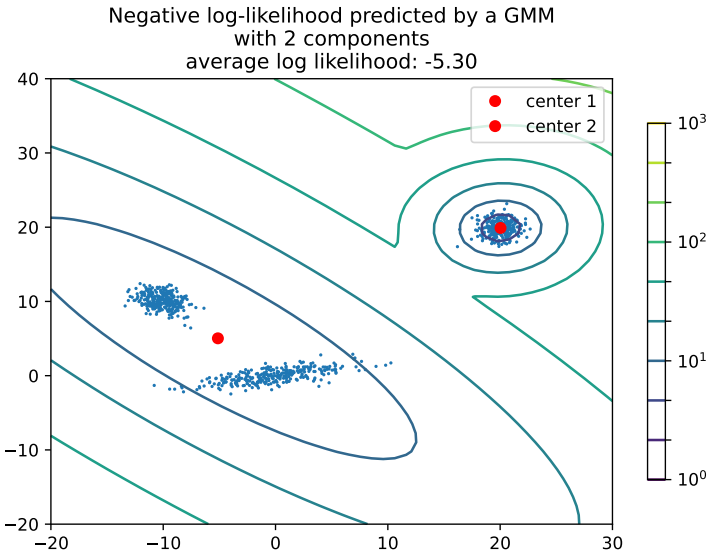


FIGURE 2 – Gaussian mixture with 2 component.

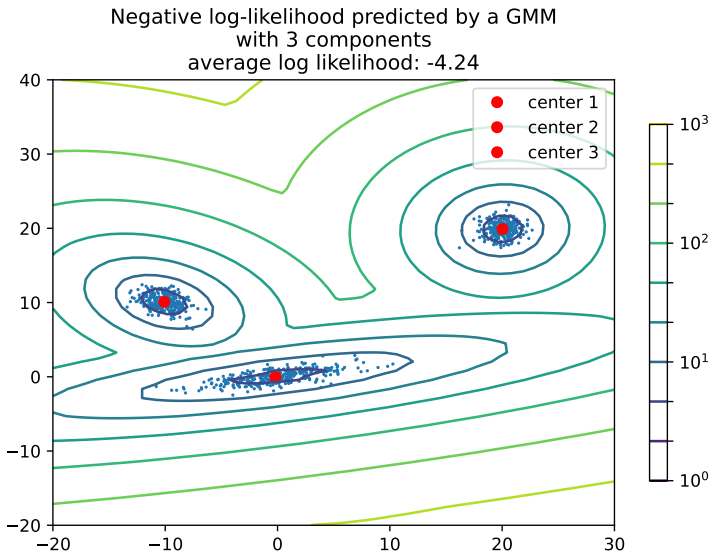


FIGURE 3 – Gaussian mixture with 3 component.

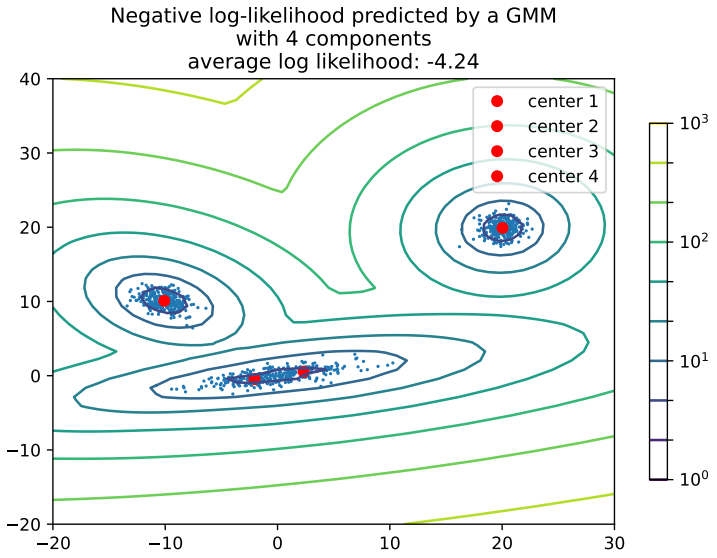


FIGURE 4 – Gaussian mixture with 4 component.

The main parameter of a Gaussian mixture is its number of components. Intuitively, it can be seen as an analogous to the number of centroids in a k-means clustering algorithm. In the previous images, we note that the average log-likelihood does not improve as much when we increase the number of components from 3 to 4, as it does when increasing it from 2 to 3. Take a few minutes to read some of the documentation on the scikit-learn page.

https://scikit-learn.org/stable/modules/mixture.html#gmm

1.2 Digits dataset

Gaussian distributions are correctly defined in any finite dimension d. Hence, we can also learn a distribution over high dimensional datasets. We will work with the digits dataset from scikit.

Estimate a Gaussian mixture distribution from the dataset, choosing the number of components that minimizes the Akaike information criterion (AIC) (in order to

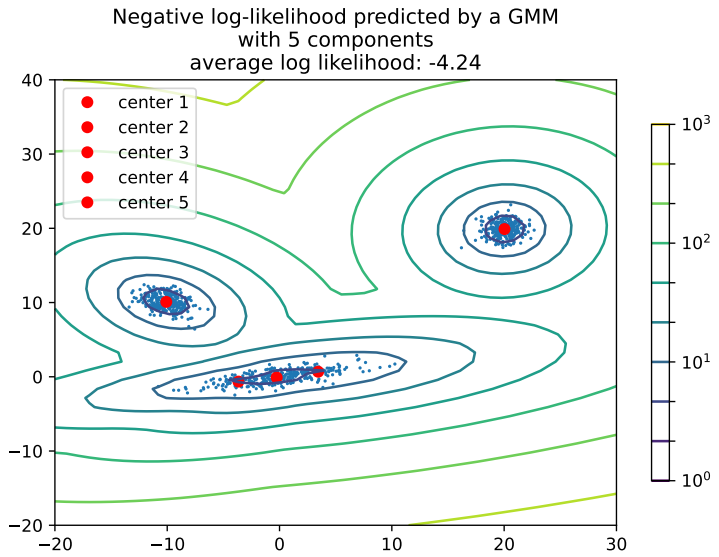


FIGURE 5 – Gaussian mixture with 5 component.

save computation time, you might try stepped numbers of components, below 200), like in figure 6. Plot the mean of each component. (see figures 7 and 8)

https://fr.wikipedia.org/wiki/Crit%C3%A8re_d%27information_d%27Akaike

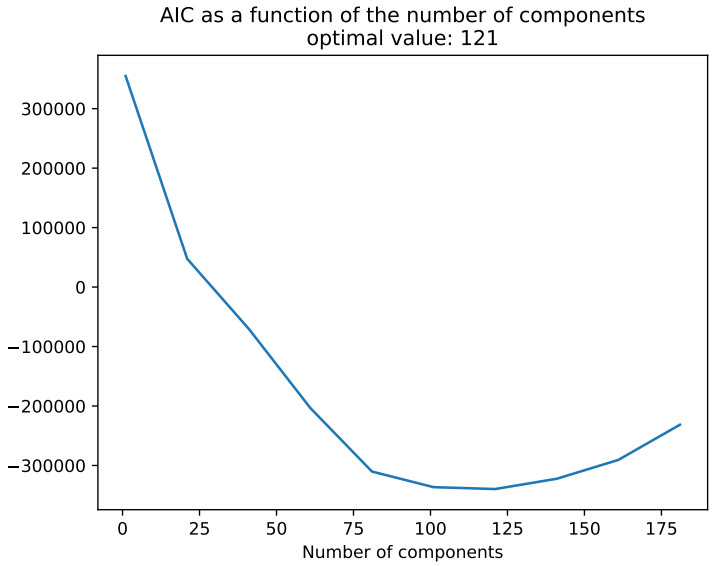


FIGURE 6 – Akaike information criterion.

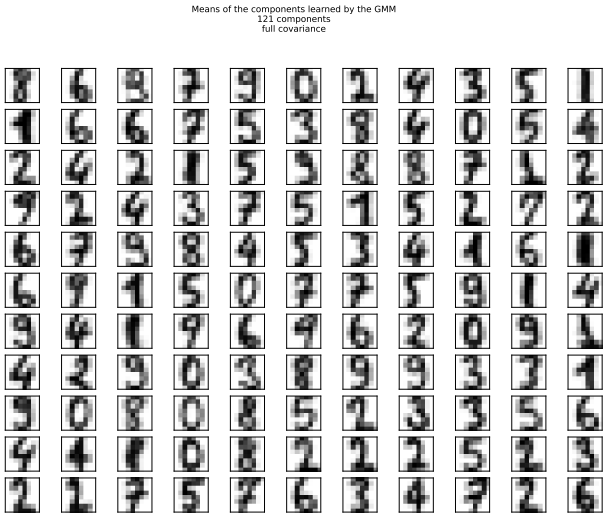


FIGURE 7 – Means of the components learned by the GMM, with 121 components and "full"" covariance (see the scikit documentation).

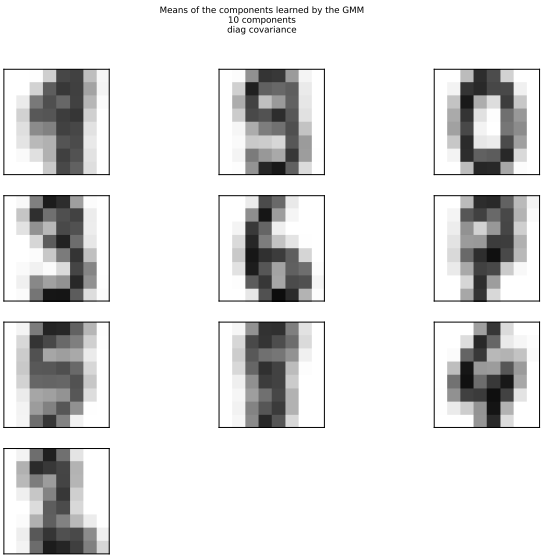


FIGURE 8 – Means of the components learned by the GMM, with 10 components and diagonal covariance (see the scikit documentation).

1.3 Generation of digits

One interesting aspect of having fitted a probabilistic model to a dataset, is that we can sample this model!

Use your previously learned Gaussian mixture model in order to generate images and compare them to images from the dataset. Explore the influence of hyperparameters (like the number of components, the covariance type) on the generated images.

An example output is shown in figure 9.

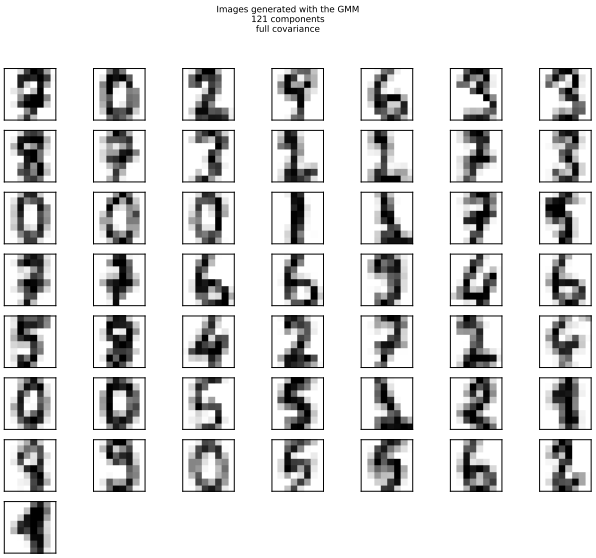


FIGURE 9 – Images generated by sampling a GMM trained on the dataset

1.4 Using Gaussian mixtures for prediction

1.4.1 *Method*

We consider a supervised learning problem, where we need to predict some output y as a function of an input x. Let us admit that we perform a density estimation on the joint variable (x, y), and obtain a distribution p.

How could we build a prediction function with p ?

1.4.2 *Application to*

We will apply this approach to the Old faithful geyser dataset.
`https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat`

The data are stored in a txt file and plotted on figure 10. By looking at the image, it seems that it should be possible to fit a GMM reasonably well to the data.

Fit a GMM to the dataset, by first finding the optimal number of components, this time according to the Bayesian information criterion (for this dataset, we expect that a number of components of 2 should be reasonable).
`https://scikit-learn.org/stable/modules/linear_model.html#aic-bic`

Then, build a predictor with this GMM and plot the prediction of the time between two eruptions as a function of the eruption time. You should observe results

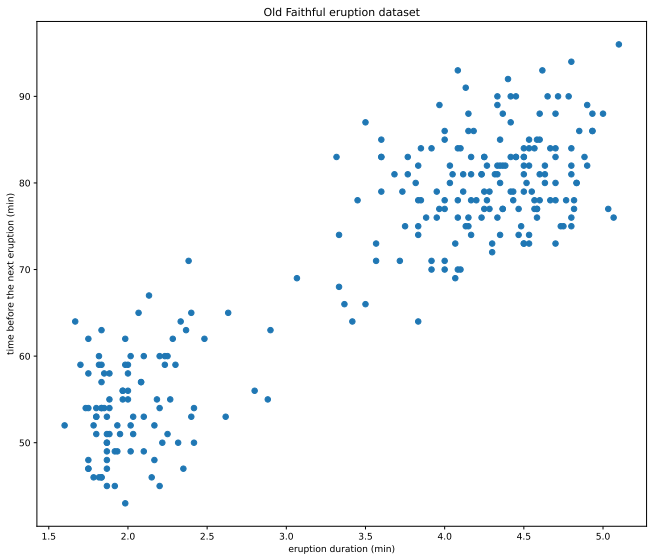


FIGURE 10 – Old faithful Geyser dataset

like figures 11 and 12.

https://en.wikipedia.org/wiki/Riemann_sum

https://www.nps.gov/yell/learn/photosmultimedia/indepth-predictingoldfaithful.htm

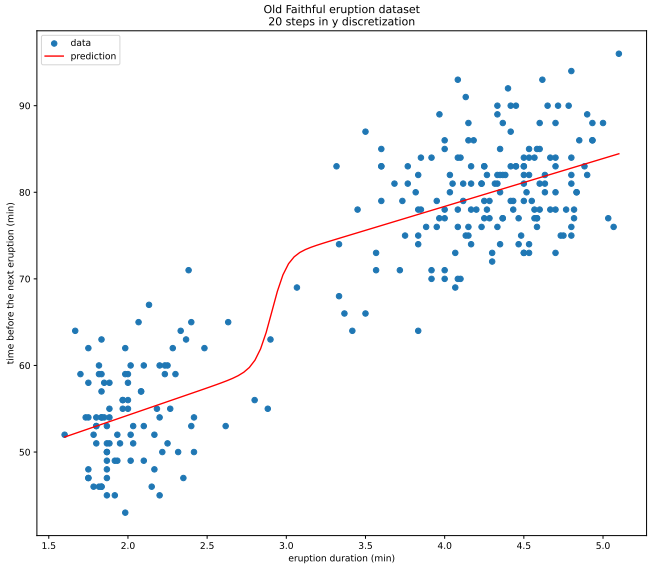


FIGURE 11 – Prediction computed from the learned GMM. In this case, 20 samples in the y space were used in order to approximate the integrals required to compute the conditional expectation of y|x.

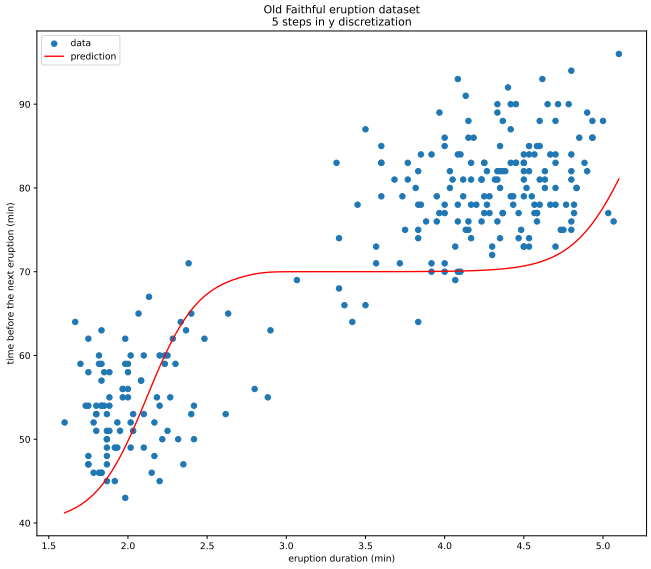


FIGURE 12 – Prediction computed from the learned GMM. In this case, 5 samples in the y space were used.

2 SIMPLICITY BIAS OF NEURAL NETWORKS

Introduction

This exercise assumes a basic knowledge of neural networks. We will witness that with some neural networks, it is unlikely to overfit the data and illustrate this with networks using the ReLU activation. In order to have some visual feedback, we will, use $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$ and the data to be learned are like the data displayed in 13.

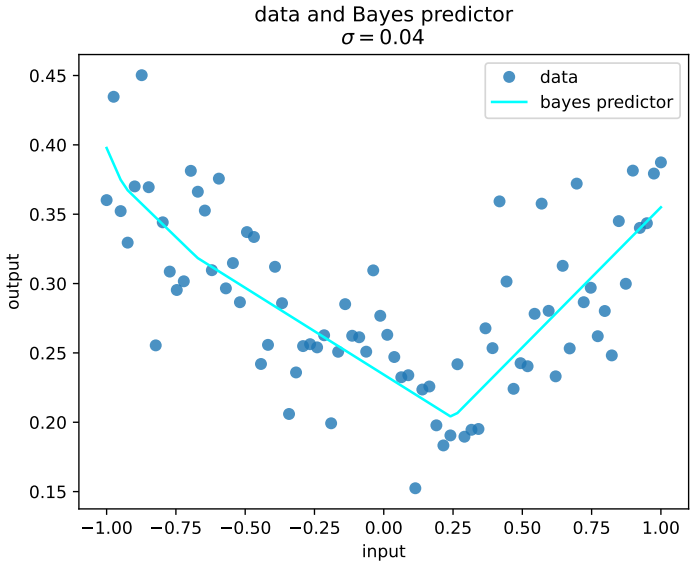


FIGURE 13 – Example target function and dataset

Let m be the number of neurons in the hidden layer, and n the number of samples in the dataset. We will see that with the specific type of networks and datasets used here, even when $m > n$ (which implies that the number of parametrers of the network is larger than n, since it is of order at least $\mathcal{O}(dm)$, d being the input dimension), no overfitting happens. This is related to the "interpolation regime", a contemporary topic in machine learning research. You can see an exmaple of such a phenomena in figure 14.

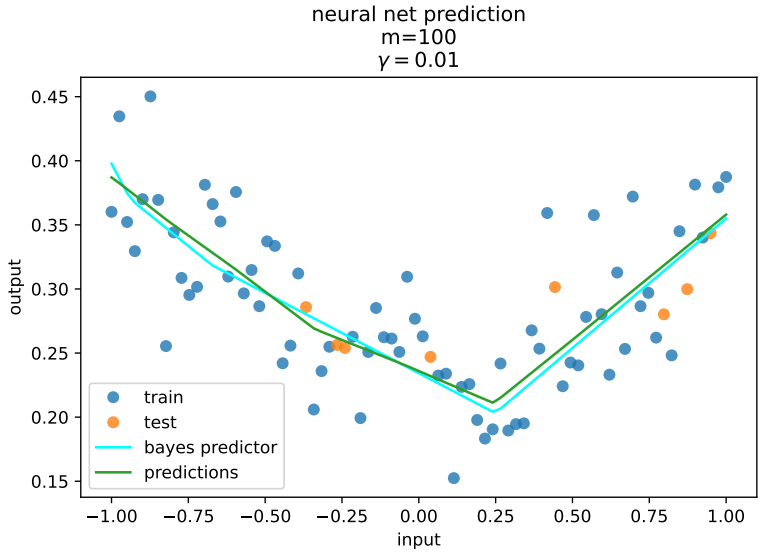


Figure 14 – Although the network has a high capacity, it does not overfit the data.

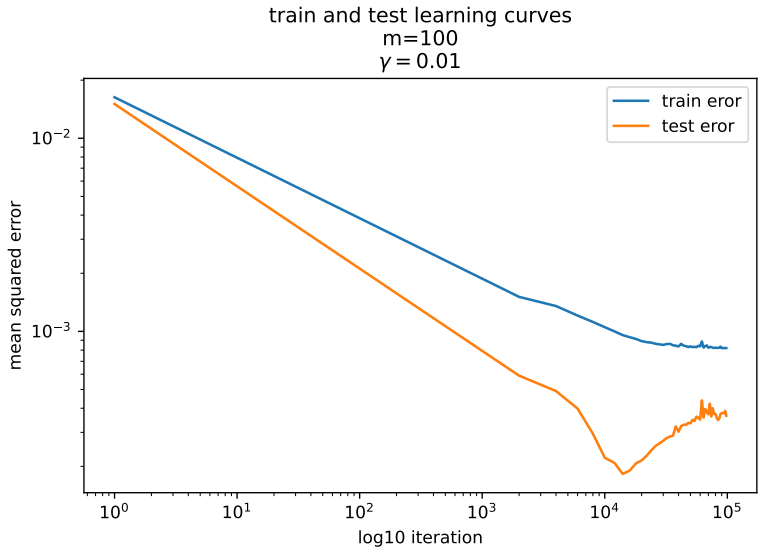


Figure 15 – Learning curves, same network as in figure 14

2.1 Definition of the neural network

We use the following architecture :

— $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \mathbb{R}$, the loss is the squared loss.

— In order to add intercepts between the input layer and the hidden layer, we add a component to the inputs $x \in \mathbb{R}^d$ and build a vector $\bar{x} = (x, 1) \in \mathbb{R}^{1,d+1}$. In our case, $d = 1$. If we take into account n samples, we add a column of 1s to the matrix $X \in \mathbb{R}^{n,d+1}$.

— The same operation is applied to the hidden layer h, in order to add an intercept before the output. We build an "extended hidden layer" $\bar{h} = (h, 1) \in \mathbb{R}^{1,m+1}$.

— The hidden layer contains m neurons. The matrix w_h containing the weigths between the input layer and the hidden layer, is in $\mathbb{R}^{d+1,m}$ (the $d+1$ comes from the intercept)

— nonlinearity : ReLU (noted σ). Also applied to the output.

— pre_h is the variable obtained before applying σ, in order to compute h. The same variable pre_y is defined for y.

— The output of the network is a single real number, hence the matrix θ containing the weights between the hidden layer and the output layer is a vector in $\mathbb{R}^{m+1,1}$ (the $m+1$ again comes from the intercept).

— Finally, the output of the neural network writes :

$$\hat{y} = f(x) = \sigma(\langle \theta, \bar{h} \rangle) \in \mathbb{R} \tag{1}$$

2.1.1 *Ressources*

Useful ressources :

— https://playground.tensorflow.org/

— https://francisbach.com/quest-for-adaptivity/

— http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

— [LeCun et al., 1998]

2.2 Implementation

Train neural networks with SGD (Stochastic gradient descent) with varying number m of neurons in the (unique) hidden layer, including $m > n$ (where n is the number of samples in the dataset) in order to observe the simplicity bias.

You will need to compute the gradients of the loss function with respect to the network parameters. There will be a gradient with respect to θ, noted $\nabla_\theta l(\theta, w_h)$, and a gradient with respect to w_h, noted $\nabla_{w_h} l(\theta, w_h)$. You will also need to edit the main SGD algorithm. For the implementation, you are free to either compute the gradients manually and write the computations in numpy, or to use automatic differentiation with libraries like torch of tensorflow.

2.2.1 *Python files*

The template files are more useful if you use the numpy / manual approach.

— **main.py** : main file, runs SGD on the neural network. You need to fix the algorithm.

— **utils.py** : computes the forward passes and the gradient computations. You need to fix the computations.

— **generate_data.py** : generates the data, you don't need to edit it.

2.2.2 Initialization

You can use the following type of initialization.

— θ is initialized uniformly in $[-\frac{1}{\sqrt{m}}, -\frac{1}{\sqrt{m}}]^{m+1}$

— Each column of w_h, that belongs to $\mathbb{R}^2$ (because $d = 1$), is initialized on the sphere of radius $\frac{1}{\sqrt{m}}$.

2.2.3 Derivatives

As the derivative of ReLU, you can use the heaviside function.
https://numpy.org/doc/stable/reference/generated/numpy.heaviside.html
https:/numpy.org/doc/stable/reference/generated/numpy.maximum.html

2.2.4 Learning rate

You will need to experiment with the hyperparameters, such as the learning rate γ in order to observe learning and the simplicity bias. As the learning algorithm and dataset generation are stochastic, you might observe different outputs.

Note that learning does not always happen, depending on the hyperparameters and dataset. In figure 16, the network has not been able to approximate the target function.

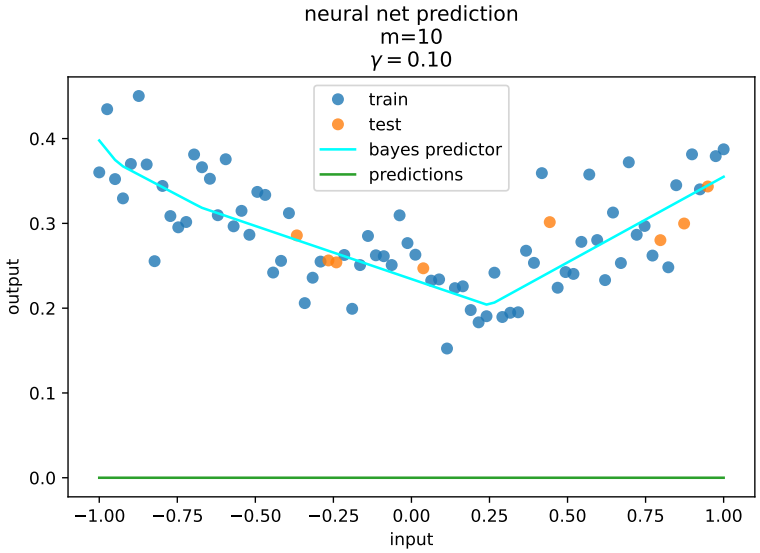


FIGURE 16 – This network has not been able to learn the data.

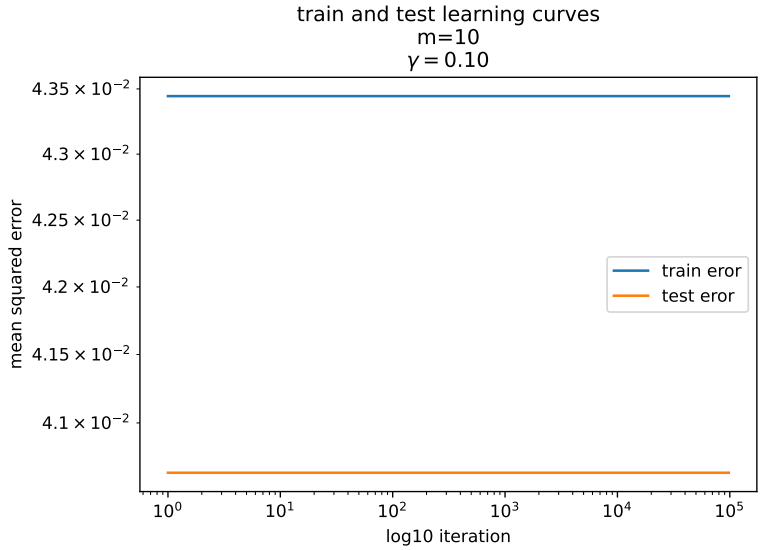


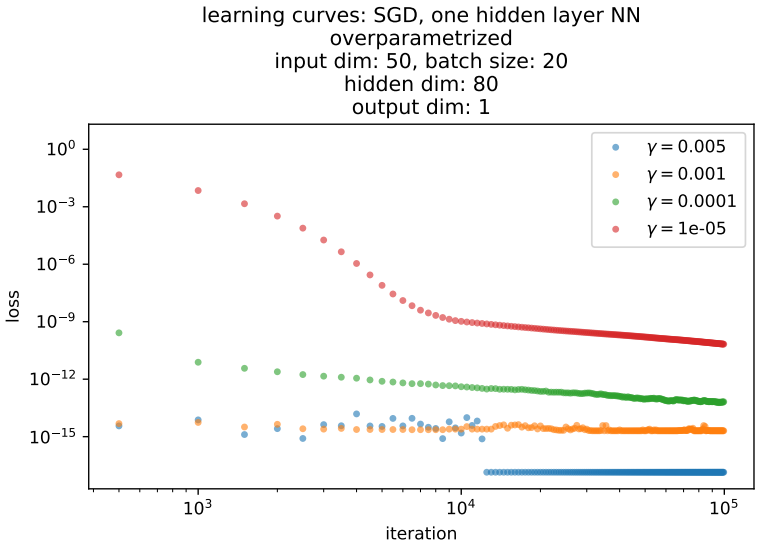
FIGURE 17 – Same network as in 16

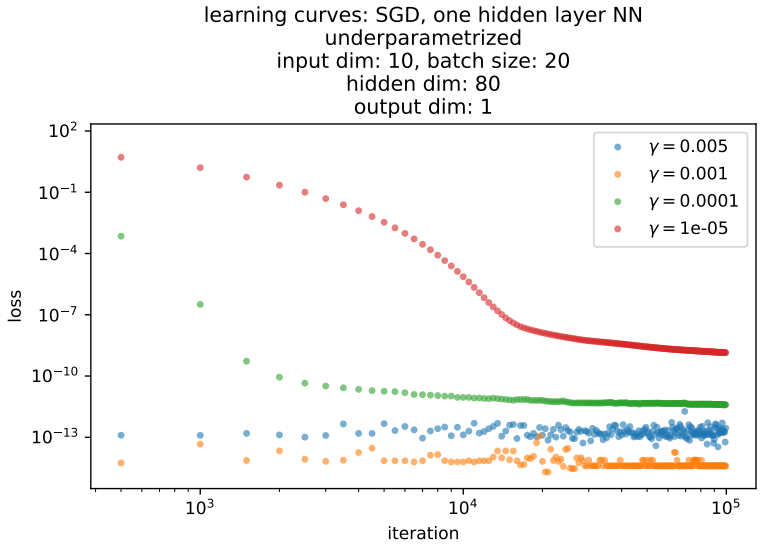
2.3 Conclusion

These neurons tend to not overfit, although some of them have a number of parameters way larger than of the minimal space containing the target function! Note that this observation is not intuitive when we have the classical interpretation of machine learning in mind, with the famous bias-variance tradeoff, which predicts that we should observe some overfitting when the number of parameters (capacity of the model) is high compared to the number of samples.

See more in this post : https://francisbach.com/quest-for-adaptivity/

3 OVERPARAMETRIZED AND UNDERPARAMETRIZED REGIMES





RÉFÉRENCES

[LeCun et al., 1998] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). Efficient BackProp. pages 9–48.